Robot Framework integration on LAVA

Contents

Introduction

The aim of this document is to provide a suitable solution for the integration of Robot Framework into the LAVA automated test infrastructure. LAVA doesn't currently support triggering or executing Robot Framework test suites. Thanks to this integration the coverage test can be extended to cover different test areas by adding additional customized libraries and toolchains.

LAVA[1] (Linaro Automation and Validation Architecture) is a continuous integration system for deploying operating systems onto physical and virtual hardware for running tests. Tests can be simple boot testing, bootloader testing and system level testing, although extra hardware may be required for some system tests. Results are tracked over time and data can be exported for further analysis.

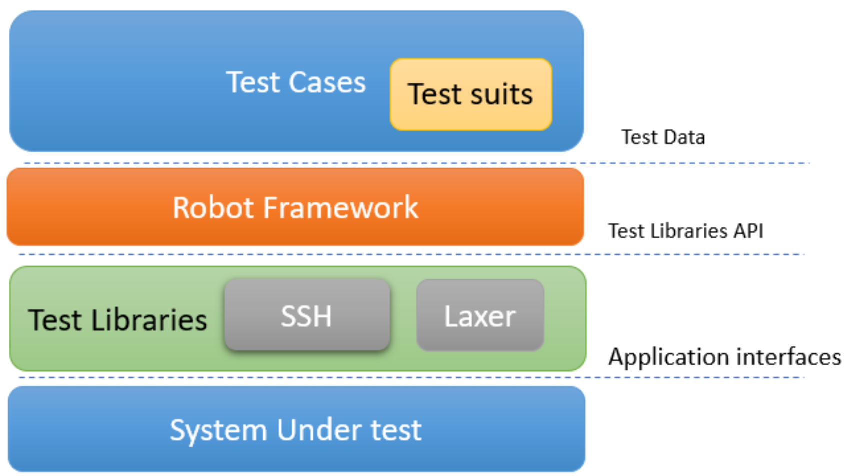
Robot Framework[2] is open source software released under the Apache License 2.0 and is a simple, yet powerful and easily extensible tool which utilizes the keyword driven testing approach. It uses a tabular syntax which enables creating test cases in a uniform way. All these features ensure that Robot Framework can

[1] https://www.lavasoftware.org/
[2] https://robotframework.org/

be quickly used to automate test cases. The best benefit with Robot Framework for the users is that there is no need for using any sort of programming language for implementing and running tests.

Integrating Robot Framework on LAVA infrastructure adds additional benefits of Robotic Process Automation (RPA), ATDD[3] (Acceptance test–driven development) and also allows the use a wide range of open source libraries developed for automation testing.

Robot framework architecture overview



Test Data

The Robot framework has a layered architecture. The top layer is the simple, powerful, and extensible keyword-driven descriptive language for testing and automation. This language resembles a natural language, is quick to develop, is easy to reuse, and is easy to extend.

Test data, the first layer of the Robot framework is in a tabular format. Since the data is in a tabular format, maintaining the data is very easy. This test data is the input to Robot Framework, once it is received, it is processed and on execution reports and logs are generated. The report is in HTML and XML format and offers detailed information about every line that is executed as a part of the test case.

[3] https://en.wikipedia.org/wiki/Acceptance_test-driven_development

Robot Framework

Robot Framework is a generic, application and technology independent framework. The primary advantage of the Robot framework is that it is agnostic of the device under test (DUT). The interaction with the layers below the framework can be done using the libraries built-in or user-created that make use of application interfaces.

Test Libraries & Test Tools

A library in a Robot Framework terminology, extends the Robot Framework language with new keywords, and provides the implementation for these new keywords. Each Robot Framework library acts as glue between the high level language and low level details of the item being tested, or of the environment in which the item to be tested is present.

Robot Framework has a rich set of built-in libraries e.g HTTP, FTP, SSH, and XML, as well as user interface and databases.

System Under Test

This is the actual DUT on which the testing activity is performed. It could either be a library or an app. Libraries act as an interface between the Robot Framework and the system under test. Hence, there is no way through which the framework can directly talk to the system under test. The Robot Framework supports various file formats namely HTML, TSV (Tab Separated Values), reST (Restructured Text), and Plain text. As per the official documentation of Robot framework, the plain text format is recommended.

Robot Framework on LAVA

There are two main constraints on automated tests setup on LAVA, the asynchronous way of updating results and user not having control over the job once it is submitted. Developers and CI pipeline can both submit jobs to LAVA, but they cannot interact with a job while it is running. The LAVA workflow defines the process of submitting a job, waiting for the job to be selected for execution, waiting for the job to complete it's execution, and downloading of the test results.

Considering the above constraints and the wide range of desired test areas, integrating the Robot Framework with LAVA provides more chances to automate complex tests by making use of its open source libraries.

The Robot Framework can add value to Apertis, but adding it to Apertis will involve developing and/or modifying Robot Framework libraries and developing a run-time compatibility layer for LAVA. The run-time compatibility layer for LAVA has two major objectives: keep testing environments as close as possible to production environments, and to adapt the execution of Robot Framework tests to suit the LAVA constraints.

Integration approach

A LAVA instance consists of two primary components **masters** and **workers** works as a [master-slave model](https://en.wikipedia.org/wiki/Master%E2%80%93slave_(technology), where the master controls one or more devices and serves as their communication hub.

The worker is responsible for running the `lava-worker` daemon to start and monitor test jobs running on the dispatcher. Each master has a worker installed by default and additional workers can be added on separate machines, known as remote workers. The admin decides how many devices are assign to each worker. In large instances, it is common for all devices to be assigned to remote workers to manage the load.

The simplest possible configuration is to run the master and worker components on a single machine, but for larger instances it can also be configured to support multiple workers controlling a larger number of attached devices in a multi node[4] model.

There are three possible approaches available to integrate Robot Framework on LAVA:

1. Integrating a standalone development setup inside the dispatcher.
2. Introduce a different device type to enable standalone docker with Robot Framework instance
3. Introducing a `test:docker` container to run a Robot Framework instance

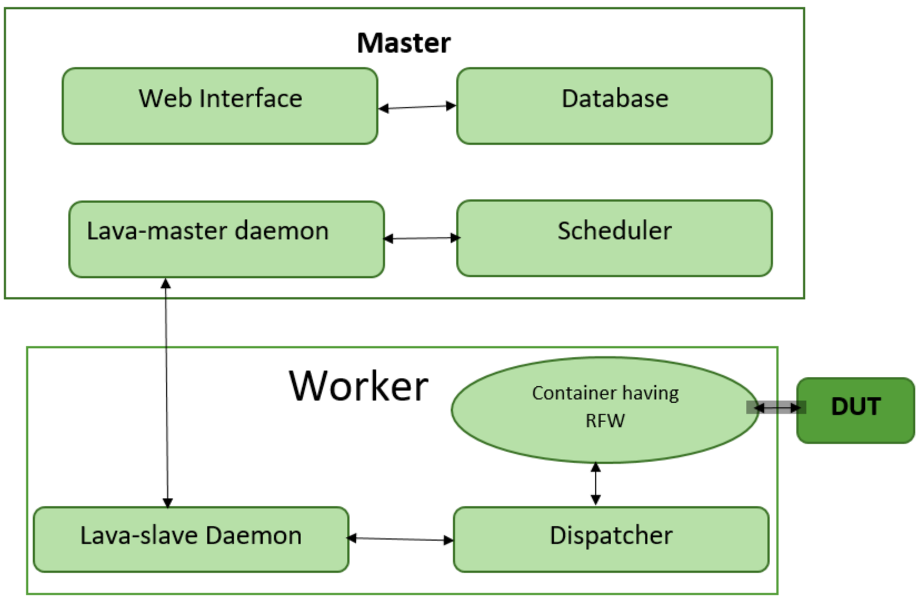
The first approach consists of creating a QEMU emulator with the Apertis SDK image and installing Robot Framework. In this approach, a user can run all automated tests related to the system and toolchain. Mainly this approach is to test the headless functionality which are part of development activities. However, running DUT related tests such as Fixed Function or HMI images is not feasible, therefore this approach is not meeting all the use cases of production readiness.

The second approach consists of creating a separate device type on the LAVA instance which contains a test Docker container where robot framework runs under the worker context. This setup provides the benefits of isolation and security, but it includes the additional effort of maintaining a different device type on LAVA. Test suites would need to specifically mention the device-type along with the architecture to run the tests on this instance. An additional advantage is that each test suite execution will be run on an independent Docker container making parallel execution possible for different jobs, this approach

[4] https://docs.lavasoftware.org/lava/multinode.html

increases the isolation of running the test suites and handling the report, but increases memory overhead if too many devices are attached and simultaneously running.

The third approach consists of introducing a `test:docker` login mechanism on the LAVA instance. This approach is completely developed and open sourced by Apertis team. Here, the job description should define the docker part by providing valid credential to pull the docker to run on dispatcher instance and execute the test steps mentioned on the test suits.



After evaluating the above three approaches, the third approach is the best fit for integrating Robot Framework on LAVA as it provides relatively easy maintenance and feature customization.

Test execution workflow

Test cases and test suites can be developed using the developers editor of choice and these tests can be run manually on the Apertis SDK, or configured to be run on LAVA.

Following workflow provide the steps to integrate Robot Framework tests and to be run on LAVA.

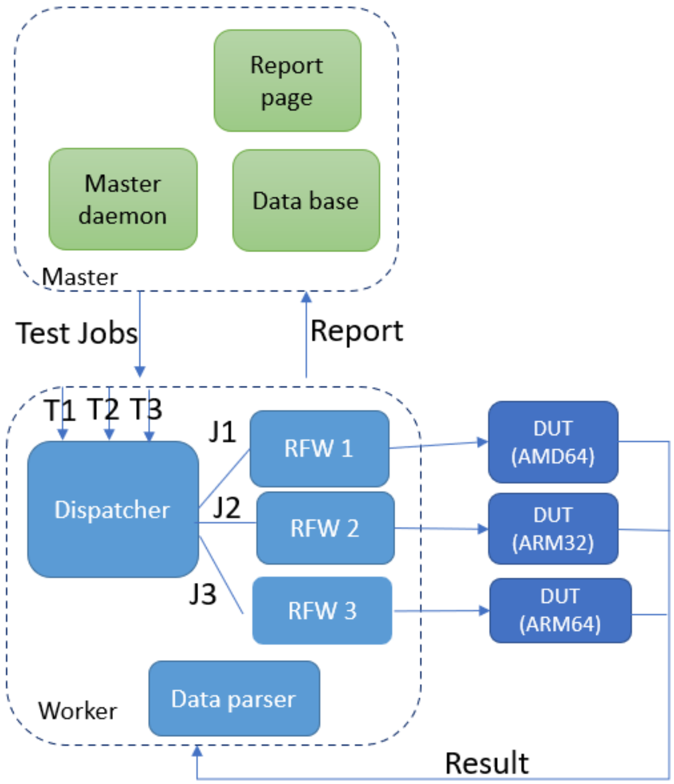
Create a common group for all the Robot Framework tests running on LAVA under apertis-test-cases[5]/lava called `group-robot-tpl.yaml` as follows:

[5] https://gitlab.apertis.org/tests/apertis-test-cases/

```
- test:
      timeout:
        minutes: 180
      namespace: rfw-test
      name: {{group}}-tests
      docker:
         image: "docker://registry.gitlab.apertis.org/infrastructure/apertis-
docker-images/{{release_version}}-rfw-docker:latest"
        login:
          registry: "registry.gitlab.apertis.org"
          user: "gitlab-ci-token"
          password: "{{ '{{job.CI_JOB_TOKEN}}' }}"
      definitions:
      - repository: https://gitlab-ci-token:{{ '{{job.CI_JOB_TOKEN}}' }}@gitlab.apertis.org/tests/apertis-
test-cases.git
          branch: 'apertis/v2023'
          history: False
          from: git
          name: robot-connman-tests
          path: test-cases/robot-connman.yaml
          parameters:
            DEVICE_IP: "$(lava-target-ip)"
            ROBOT_FRAMEWORK_CONNMAN_URL: |-
         https://gitlab-ci-token:{{ '{{job.CI_JOB_TOKEN}}' }}@gitlab.apertis.org/tests/robotframework.git
```

This template provides the basic information and credentials for fetching and running the Robot Framework with Docker in LAVA. Tests can be added utilising this template.

Framework operation



The above diagram shows the basic workflow of LAVA jobs using Robot Framework. A job will be created on the master daemon which specifies a suite of tests (T1 to T3), which DUT that the tests will be run on and the Apertis release plus image type which they should be run against.

When it is time for it to run, the master daemon passes the job to the dispatcher with the required DUT. The dispatcher will launch a Robot Framework docker instance (J1 to J3) which will connect to the required DUT using the SCP and SSH protocols to copy required files to and from the DUT and execute commands on it, rather than copying the entire test suite and Robot Framework to the DUT and executing it from there. This has the advantage that minimal alterations will be made to the image that is being tested. The required test suite will be executed from within its docker environment, with each job running in its own fresh isolated docker environment, ensuring that it is not affected by content left from previous jobs.

Once the test execution is completed, Robot Framework will generate a test report and a number of logs which will be copied from the docker instance and shared with the LAVA server. Once this is done the docker instance will be cleaned up. A summary of the testing results and the test reports/logs will be made available via the dashboard[6].

It is likely that Robot Framework tests will have dependencies which are required for the tests to run correctly. Where these dependencies form part of the test harness in the docker instance (for example, libraries to drive peripherals such as a touch simulator to simulate touch events for HMI tests), these should form part of the docker definition and installed from the Apertis repositories when the docker instance is created. Where these dependencies need to be available on the DUT, they either need to be preinstalled as part of the image or are required to be added to the image during testing (such as by applying an overlay on OSTree based images).

When run, the Robot Framework generates three files in its output directory:

- `output.xml`: An XML formatted record of the test execution, including data such as test names, statuses, messages, and tags.
- `log.html`: A detailed HTML formatted log of your test execution, which includes timestamps, keywords, arguments, screenshots, and console output.
- `report.html`: An HTML formatted summary of your test execution, which shows the overall statistics, test cases run and errors raised.

Currently the LAVA server is not processing any of these Robot Framework test reports, it only tracks the test status. We plan to add a data parser and provide the parsed data to LAVA. The Robot Framework reports will also be stored and a link provided to them from the LAVA report.

The Robot Framework only generates the status report at the end of test execution. To allow for more real time tracking of the testing, the Robot Framework provides a listener mechanism which can be used to provide fine grain monitoring of each individual tests execution. A listener script should be written to interface between the Robot Framework and LAVA and made available as part of the main test scripts. This integration will provide greater integration between the Robot Framework and the existing LAVA infrastructure and will be very beneficial when debugging failing tests.

Impact analysis on Apertis distribution

Infrastructure

Integrating Robot Framework on existing Apertis infrastructure will requires the following changes :

[6] https://qa.apertis.org/

- Improvement of LAVA workers to enable them to run docker instances.
- Configure pipelines to ensure the capture of all the Robot Framework results.
- Extend the Apertis test report site[7] to show the Robot Framework results

Development environment

The current development environment integrates Robot Framework with all its standard libraries, along with the `SSH` library as part of the SDK distribution. Using the Apertis SDK a developer can write Robot Framework test cases to run on the SDK and DUTs running Fixed Function or HMI images.

Test development

- Impact on Apertis development is that we have start developing new test suites for robot framework.
- Start developing new yaml files which helps in executing the robot test suites from containers
- Apertis tests needs to rewrite the existing LAVA test job to execute the robot test suites

Testing

- With approaches mentioned above we can keep the existing scripts as they are and start executing tests defined with the new Robot Framework test suites which will help to improve the test coverage.

Summary

The integration of the Robot Framework into the Apertis, enables tests to be written using this simple, yet powerful and easily extensible testing framework for Apertis whilst also taking advantage to the many features provided by the Apertis test framework:

- End to End workflow
  - LAVA pipelines can take care of all test stages: control of board power; flashing & booting images; loading tests; running the tests; and reporting test results.
  - Tests can be run in parallel on different targets, reducing test cycle time, when compared with manual tests run by a limited test team.
  - Devices can be reserved for specific tests or specific users.
- Internet facing Web Service

[7] https://qa.apertis.org

    - One centrally hosted and maintained front end service which can be utilised by multiple teams, each providing worker systems connected to their specific DUTs.
    - Internet connectivity enables collaboration with external partners.
    - Remote management of devices. Many maintenance tasks can be completed without physical access to DUTs.
    - Remote access to users for running tests, viewing logs & reports.
    - Role based access permissions allowing granular control over access to functionality and specific DUTs.
    - Access to mail notifications and alerts.
- Sharing of physical assets between multiple software projects
    - DUTs can be shared between multiple projects, such as teams focusing on different operating systems or teams focusing on different software stacks within a larger operating system can schedule jobs to be run on shared hardware, reducing the number of physical devices needed for testing across an organisation
- Continuous testing
    - Periodic triggering of test runs against DUTs as part of continuous testing to ensure acceptable operation as system evolves.
    - Reuse of common tests between integration and continuous testing regimes avoiding duplication of effort.
- Handles inconsistency.
    - Retry mechanisms mitigate against test failures due to temporary failure of ancillary operations, such as transient download failures.
- Inbuilt reporting dashboard
    - Insight full metrics available in the inbuilt dashboard.
    - Access to full test reports and test definitions.
    - Access to test logs including timing metrics.